Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2022) 30: 328 – 343

doi:10.3906/elk-2103-110

# Performance analysis of lightweight Internet of things devices on blockchain networks

**Cem KÖSEMEN**[1*], **Gökhan DALKILIÇ**[2], **Şafak ÖKSÜZER**[1]
[1]Department of Computer Engineering, The Graduate School of Natural and Applied Sciences, Dokuz Eylül University, İzmir, Turkey
[2]Department of Computer Engineering, Faculty of Engineering, Dokuz Eylül University, İzmir, Turkey

**Received:** 23.03.2021 • **Accepted/Published Online:** 25.12.2021 • **Final Version:** 04.02.2022

**Abstract:** Potential integration or cooperation of the Internet of things (IoT) systems and the blockchain technology is nowadays attracting remarkable interest from the researchers. These inter-operating systems often have to rely on low-cost, low-power, and robust IoT devices that can communicate with the blockchain network through smart contracts. In this work, we designed and ran a benchmark study for ESP32-based lightweight IoT devices interacting within the Quorum blockchain. A software library was built for ESP32 devices to enable elliptic-curve digital signing, Keccak-256 hashing, decoding, encoding, and secure private key generation capabilities, which all are the basic functional requirements for running a blockchain client. The running times and power consumption values of these essential operations were analyzed and discussed in detail. We also deployed two smart contract functions on the Quorum network and analyzed the performance of the IoT device while interacting via these methods. Further optimizations were proposed and implemented to speed up the processes and save energy, which resulted in a 2.5x increase in the transaction posting speeds and a 2/3 decrease in the energy costs. Another performance comparison between an ESP32 and a PC client was also provided with.

**Key words:** Blockchain, Internet of things, ESP32, smart contracts

## 1. Introduction

The Internet of things (IoT) technology maintains its importance with billions of connected devices, where the number of devices increases each year [1]. IoT is basically a network of all kinds of interconnected devices like microcontroller units, smartphones, wearable devices, sensor devices, etc. Although IoT devices have data processing and network communication capabilities, they are mostly lightweight devices with much lower computing resources than general-purpose personal computers (PCs).

Relying on centralized server-client architectures increases the possibility and severity of security breaches. So, recent research and industrial work focus on integrating blockchain technology with these centralized systems. Blockchain enables the connectivity of IoT devices in a decentralized and efficient way since there is no single point of failure and bottleneck problem. Also, the cost of the centralized systems will have difficulties in the near future while dealing with the exponentially increasing number of IoT devices [2].

IoT devices often autonomously communicate through blockchain smart contract transactions. These

*Correspondence: cem.kosemen@ceng.deu.edu.tr

smart contracts provide secure information exchange between the devices over the Internet. Blockchain also offers immutability of data, which can be used in IoT identity and access management systems. Since IoT devices mostly work without human interaction, smart contracts provide a convenient solution.

Although basic concepts behind blockchain existed in the early 1990s, it was introduced as a whole system in 2009 by Bitcoin [3, 4]. Bitcoin proposed a decentralized autonomous electronic cash system without a single point of failure. With Bitcoin, people realize that blockchain technology can create a reliable basis, where different parties who do not trust each other can communicate securely. All transaction records are stored in blocks, where the previous block's hash value is also stored in the next blocks. These stored hash values provide immutability for the stored data.

Adding a new block to the blockchain should be approved autonomously for all parties to agree. This process is provided by an algorithm called the consensus algorithm. Since blockchain technology is essentially a peer-to-peer (P2P) network, participants keep a copy of the blockchain as a node and can participate in the consensus algorithm. The consensus algorithm for Bitcoin is a proof of work (PoW) algorithm. PoW is based on solving a mathematical puzzle. The first participant to solve that puzzle can add a new block to the chain and earn Bitcoin for the energy it consumes.

Blockchain technology has become very popular after Bitcoin made autonomous money exchange possible without a central authority. Subsequently, the concept of a programmable blockchain emerged with Ethereum [5]. With Ethereum, it has become possible not only to transfer money but also to store the transaction record of any data change on the blockchain. Ethereum actually developed a distributed computing concept by running code pieces called smart contracts on the Ethereum virtual machine (EVM). Ethereum uses a simpler PoW algorithm than Bitcoin as its consensus algorithm.

After Bitcoin and Ethereum, blockchain technology is used in other fields like education and energy with more emphasis on data immutability. New approaches for blockchain systems were designed for specific requirements in different cases. According to the visibility of the data, we can divide the blockchain into two different types: public and private. According to the method for joining the blockchain network, there are two main types of blockchains: permissioned and permissionless [4].

If the data is accessible by everyone, it is called a public blockchain. If the data can be accessed with permission from a certain authority, it is called a private blockchain [6]. Public blockchains do not provide complete privacy. Therefore, the fact that the data is visible to everyone is not very suitable for some cases.

Permissioned and permissionless methods are determined according to the existence of a procedure that requires permission from a specific authority to participate in the network. If a procedure requires permission to join the network, it is called permissioned; otherwise, it is permissionless. Permissioned systems provide a control mechanism where peers participating in the network can be managed [4].

Bitcoin and Ethereum are examples of public permissionless blockchain structures [6]. Quorum and Hyperledger Fabric are examples of private permissioned blockchain structures. The consensus algorithms used also vary according to the blockchain types. In public permissionless systems where everyone can participate, algorithms such as PoW that grants awards are often preferred. However, these algorithms are slow and inefficient for permissioned systems [7]. The speed of the blockchain does not directly depend on whether being public or private; rather, it depends on the consensus algorithm used.

In private permissioned blockchain systems, speed and efficiency are more important. The participants can be assumed as reliable, so the system should work as fast as possible. In these systems, consensus algorithms such as Raft or Istanbul Byzantine fault tolerance (IBFT) are preferred [8, 9]. Depending on the reliability of

the network's participants, either slow consensus algorithms that require more power or more efficient and fast consensus algorithms can be preferred.

Quorum (https://github.com/ConsenSys/quorum) is a permissioned blockchain framework, which was introduced by JP Morgan [10]. Quorum is open-source and based on Ethereum, but Quorum uses different consensus algorithms (Raft and IBFT) and provides private transactions and contracts. Quorum also does not use the transaction fee method of Ethereum, and Quorum is generally used in enterprise-level services.

Most of the IoT devices currently used have constrained computing capabilities with low computation and memory resources. Also, IoT devices should keep their power consumption minimal and use their resources effectively for optimized battery life. ESP32 is one of the most popular and successful examples of these low-cost & low-power devices [11]. ESP32 has sufficient computation power for most IoT tasks with built-in robust Wi-Fi and Bluetooth communication module. It also has a cryptographic acceleration module that is very useful for providing solutions to privacy requirements in IoT.

In the blockchain, the data is tamper-proof. Someone may try to change the data, but he cannot run away without being caught. So, in our system, the data is tamper-proof, and as Quorum is a private permissioned blockchain, the data is anonymous. In our system, we used public transactions. Besides Ethereum's public transaction, Quorum has added the private transaction concept. In a private transaction, only a group of nodes (for example, the nodes in a single department) can reach the unencrypted data. For all the others, the data is confidential. Blockchain, especially private permissioned blockchain, was selected to store data in our project for these security services.

Per motivation, since the integration of IoT systems with blockchain technology increases, we need robust and reliable IoT devices for those systems. A low-cost and low-power IoT device with networking and cryptographic capabilities is an essential requirement for those systems. So, we decided to provide an experimental benchmark study for our selected device, ESP32.

Our main contributions are;

- We inspected lightweight IoT device performance in a real-world scenario using ESP32 IoT device and Quorum blockchain network with smart contracts. Also, we compared it with a regular PC's performance. For performance metrics, we calculated the time required for specific jobs along with the power required to execute these jobs.

- We implemented a blockchain client library for ESP32. This software library provides Keccak-256 hashing and elliptic-curve digital signature algorithm (ECDSA) functionalities, also with required encoding and decoding capabilities.

- We offered different solutions to optimize the IoT device's performance while interacting with the blockchain network as a client. We developed a client that can communicate with the smart contract by only storing the public key in ESP32. This IoT device does not depend on any other gateway device, and it can send transactions and make smart contract method calls only by itself. It can interact with deployed smart contracts on the blockchain network.

## 2. Related works

Recently, integrating blockchain and IoT devices got the attention of researchers. This combination offers big advantages on privacy and decentralization. Lin et al.'s work [12] uses blockchain for providing security and

integrity of data on food traceability and smart agriculture IoT systems. They provide a solution to trust issues and human intervention using automated smart contracts. All entities in the IoT system can access the immutable data stored in the blockchain. Also, IoT clients as light nodes can send transactions to blockchain network through IoT gateways.

Cha et al.'s work [13] focuses on solving privacy leakage using blockchain. Blockchain network stores and manages users' privacy preferences and device information via Ethereum smart contracts. IoT devices can access the blockchain network using a gateway device. Nayak et al.'s work [14] presents a system that uses smart contracts for providing authentication and authorization for cloud tenants. They use Quorum private blockchain platform as similar to our work. Quorum blockchain is selected for its scalability and resilience advantages over other public blockchain platforms.

Novo's work [15] offers a decentralized way for access management in IoT networks. Access information of the IoT devices is stored in a public blockchain that uses PoW consensus. A single constant smart contract defines all the access control permissions. IoT devices are used as light nodes in that work too. They can send transaction requests to gateways. Ma et al.'s work [16] uses a public blockchain network as a key management center. Blockchain provides more decentralization and scalability than cloud services.

Another most common consensus algorithm besides PoW is proof of stake (PoS), where each stakeholder uses her power of the number of stakes she has, without spending work power. Like PoW, PoS is also used in permissionless blockchains. Algorand protocol [17] is an example of pure PoS. In regular PoS, the stakeholder's stake is blocked throughout the operation. That is why the stakeholder will not use his or her total money. However, in Algorand, all the money can be used without hesitation. Another kind of PoS is delegated proof of stake (DPoS), where a group of stakeholders is delegated throughout the process [17].

ESP32 offers a solution to IoT system developers as a robust and low-cost IoT device for various applications. Technical reference manuals and datasheets of ESP32 can be found on Espressif System's website (https://www.espressif.com). Ghosh et al. built an ESP32 based smart IoT system for saline level monitoring in water using message queue telemetry transport (MQTT) [18]. Abdullah et al.'s project monitors gas leakages using ESP32 devices [19]. There are also smart surveillance systems and real-time photovoltaic monitors developed with ESP32 [20, 21].

We also examined works that evaluated the performance of private blockchain platforms. In Baliga et al.'s work [22], they analyze the throughput and latency values of the Quorum blockchain. Throughput means several transactions processed by the blockchain network. Latency in the blockchain network shows the time difference between the transaction request sent and the response received by the client. Raft performs better than IBFT when throughput is high (over 1650 transactions per s). However, IBFT outperforms Raft in lower throughput values. Also, they showed that IBFT has higher latency values than Raft.

Compared with Corda that is also Raft based permissioned and private blockchain platform, Quorum has a much higher throughput and is better in privacy/confidentiality [23]. Also, enterprise Ethereum Alliance (EEA) is a permissioned blockchain [24] that is Ethereum based offering BFT consensus algorithm to be used.

In Nasir et al.'s work [25], they run a performance and scalability analysis on the Hyperledger Fabric blockchain platform. They showed that a private blockchain platform can achieve very high throughput values with a different number of peers.

## 3. Materials and methods

In this section, we explained the required tools and methods for developing a robust and fast IoT lightweight client. First, we explained the IoT device that is used in this work, ESP32. Afterward, the main cryptographic requirements to communicate with a blockchain network are explained in detail. Furthermore, Quorum private blockchain platform is examined with its consensus protocols and smart contracts.

### 3.1. ESP32 IoT device

ESP32, which is also the chip's name, is an IoT device produced by Espressif Systems. ESP32 provides a capable and robust Wi-Fi interface. Additionally, these boards are low-cost, low-power, and easy to use. We used the ESP32-WROOM-32 version of ESP32 in our implementation and benchmark tests. It has a 32-bit dual-core microprocessor with up to 240 MHz clock frequency. It has 520 KB static random-access memory (SRAM) for data and instructions and 448 KB read-only memory (ROM) for booting and core functions. It supports Wi-Fi 802.11 b/g/n, and Bluetooth 4.2 basic rate/enhanced data rate (BR/EDR), and Bluetooth low energy (BLE) [11].

ESP32 also has features for security applications like secure boot and flash encryption. It has hardware acceleration for commonly used cryptographic functions like advanced encryption standard (AES), secure hash algorithm (SHA-256), Rivest–Shamir–Adleman (RSA), elliptic-curve cryptography (ECC), and random number generator (RNG). ESP32 uses FreeRTOS operating system to support real-time applications.

We used ESP32s as clients, which means they can interact with the blockchain network, but they do not store the blockchain data like the full nodes in the network. Also, we used the LM35 temperature sensor as a data acquisition module simulator. LM35 outputs the environmental temperature with its output voltage. ESP32 communicates with the smart contracts using the data it obtained from LM35. Calling too many smart contracts from the clients (ESP32) may be a headache for such IoT devices. Every smart contract call needs some memory space that is one of the scarce resources of an IoT device. As these devices are cheap, several clients can be used at the same time, each calling a few smart contracts.

### 3.2. Hashing, signing and private key generation

A cryptographic hash function is one of the essential parts of a blockchain system. Quorum uses the Keccak-256 function for its hashing operations [26]. Keccak-256 is the winner of the SHA-3 competition and was selected as the standard by the American National Institute of Standards and Technology (NIST).

Espressif's official ESP32 library has already implemented the hardware-accelerated SHA-256 function. However, ESP32's library does not support Keccak-256 yet. Since we must strictly use Keccak-256 for Quorum, we used C language software implementation of Keccak-256.

Public key cryptography is an essential requirement in blockchain networks. Clients have a public key, also called the account address. The account address is known publicly and used for identifying the client. The private key is generated randomly when a new blockchain account is created. A private key is created and stored in the device, and it is never shared with other devices or the blockchain network.

The client device must sign the transaction information with its private key using an elliptic-curve digital signature algorithm (ECDSA) to obtain a signed request. ECDSA is the upgraded version of the digital signature algorithm (DSA) using elliptic-curve cryptography. According to NIST publication [27], the acceptable key size for digital signature generation using DSA is 2048 bits, and the corresponding number in ECDSA is 224 bits. So,

using ECDSA compared to DSA is advantageous. The generated digital signature provides authentication and message integrity before the transaction is accepted in the Quorum network. Quorum uses secp256k1 elliptic curve (EC) that is a NIST standard [28, 29].

ECC methods are more suitable than RSA for lightweight IoT devices. Because ECC achieves the same security level of RSA with lower-key sizes, which reduces the computational complexity of the algorithm [30]. According to Bitcoin Core's implementation (https://github.com/bitcoin-core/secp256k1), secp256k1 EC is constructed especially for efficient computation. According to Bitcoin's documentation, if it is optimized properly, secp256k1 EC is 30% faster than other ECs.

When a Quorum account is created, the account holder device creates a 256-bit private key with random number generation methods. This private key should never be exposed outside and must stay in the device securely. The public key, also called the account address, is generated using that private key with ECDSA.

RNGs must have a good entropy source to produce high-quality randomness, which means it should not be predictable [31, 32]. Creating a private key is a random number generation task, so we need a cryptographically secure RNG and an entropy source in the IoT device. ESP32 has a true random number generator (TRNG) that can be used in cryptographic applications. TRNGs often use physical events as their entropy sources. In ESP32, thermal noise and asynchronous clock mismatch are used as entropy sources.

### 3.3. Quorum blockchain platform and smart contracts

We deployed a basic Quorum blockchain network in a virtual machine that is provided by Google Cloud platform. This blockchain network has three full nodes. Quorum blockchain provides a private and permissioned blockchain structure. We can create private contracts that only a specific set of users can use. Also, only authorized peers can join this network. In Quorum, smart contracts are implemented using Solidity that is a high-level object-oriented programming language.

Quorum is an open-source enterprise blockchain infrastructure based on Ethereum [10, 33]. Quorum provides Raft and IBFT as its consensus protocols. The biggest difference from classical Ethereum infrastructure is that Quorum can provide high performance without the need for any cryptocurrency system. Quorum maintains privacy and confidentiality on a closed blockchain network. Private transactions can proceed in Quorum so, privacy and confidentiality management can be done between different parties on the blockchain network. One of the biggest advantages provided by Quorum is that it supports tools developed for Ethereum, like Metamask, as it works with the Ethereum infrastructure. In this study, Quorum is selected instead of classical Ethereum because for high throughput, Raft and IBFT are used in Quorum, and they are not available in classical Ethereum. And also, being a public blockchain, Ethereum uses a cumbersome consensus algorithm that consumes too much power.

The Raft consensus protocol is customized to meet the needs of an enterprise blockchain infrastructure [8]. Since it is a vote-based consensus algorithm, nodes in the network must choose a leader node by voting. There must be at least three nodes in the network, as the majority must be provided for the leader node. A result with an equal number of votes renders the system dysfunctional. Raft achieves very fast results compared to other known consensus algorithms. Unlike PoW, it works on permissioned and private blockchain networks where the goal is to get a faster consensus [10].

IBFT consensus protocol can be used where the users are authorized users, which means the network is secure compared to BFT that was developed for insecure networks. Compared to BFT, IBFT makes less broadcast which creates minimum communication cost between the peers. Besides that, compared to BFT,

IBFT is much more flexible, and peers can be added to the network or removed from the network [9].

### 3.4. Web3 remote procedure calls

Remote procedure call (RPC) library of Web3 application programming interface (API) is used to interact with Ethereum network. RPC can also be used for interacting with the Quorum network and deploying smart contracts. Figure 1 represents the basic network data flow of sending a raw transaction using RPC. Much more details of the network data flow are present in [5]. We used hypertext transfer protocol (HTTP) as the communication protocol.

First, an *eth_getTransactionCount* RPC request is performed by the ESP32 client. As a response from the Quorum network, the client receives a nonce that is going to be used later in the transaction sending process. Transaction count request is represented in step 1 of Figure 1. After that, the client sends a *web3_sha3* RPC request with the name of the contract method that is wanted to be used. The Keccak-256 hash of the sent data, also called as contract application binary interface (ABI), is received by the client as the response, which can be seen in step 2 of Figure 1.

As shown in step 3 of Figure 1, obtained contract ABI is encoded in a recursive length prefix (RLP) header by the client with other essential parameters. After that, it is sent within a *web3_sha3* RPC request. The hash of that RLP header is signed with an ECDSA using the client's private key and sent within a *eth_sendRawTransaction* RPC request. The resulting transaction hash is returned from the Quorum network as a receipt of that request, as shown in step 4 of Figure 1.

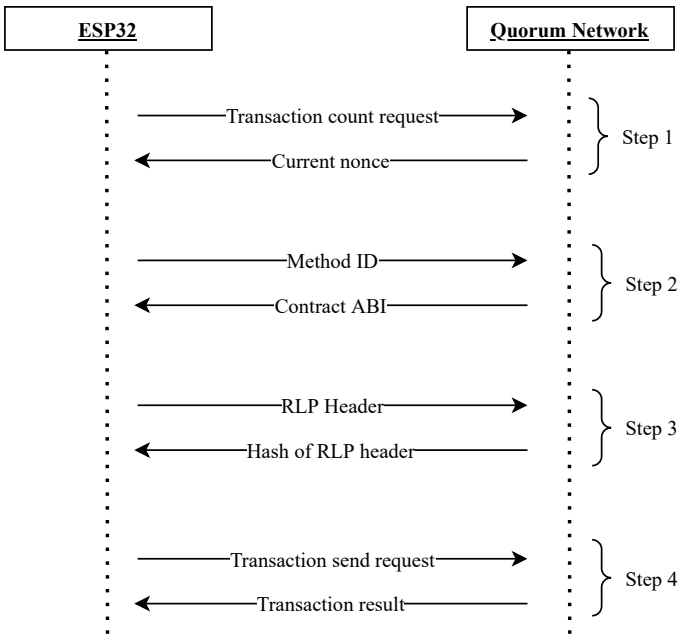


**Figure 1**. Network flow of transaction send operation.

## 4. Performance evaluation and discussion

To test the performance of an IoT system interacting with the Quorum blockchain (as of both an ESP32 device and a PC), we implemented a custom smart contract containing two unique functions. The first function,

named as *saveTemperature*, takes a numeric value as the input and stores it in the blockchain network. This function is used for recording values obtained from the LM35 temperature sensor connected to ESP32 (in fact, representing any lightweight IoT client). Second function, named as *getLastTemperatures*, is used to test the read-only call function. It returns the last three temperature values sent from a valid account (e.g., an IoT client). Call functions do not require validation or creating a transaction in the blockchain network; they only return the current value of a transaction. We examined the performance of send and call operations on valid transactions, hashing, and ECDSA, as well as the quality of randomness and power consumption values.

### 4.1. Performance of transaction sending

The transaction send operation starts with a *eth_getTransactionCount* call, as we can see in Figure 1. However, *eth_getTransactionCount* request is not necessary for every transaction sent. After booting up, the client device can get the nonce value for once using *eth_getTransactionCount* RPC. Later on, the obtained value from that request is maintained as a nonce and incremented by the client in each subsequent send request. As the first improvement, we avoided sending that request in every round. This allowed us to make one less HTTP request on each transaction send operation. So, we do not have to repeat step 1 of Figure 1 on every transaction send request, because the nonce value can be incremented in the device. This increment is important as nonce values must be used in an incremental order without missing any value. The nonce value is stored in the local memory of ESP32 and incremented one by one before every transaction. The nonce value is used to eliminate double-spending.

As the second improvement, we avoided sending two extra *web3_sha3* requests by implementing the Keccak-256 hash function on ESP32. This improvement eliminates the network traffic on 2$^{nd}$ and 3$^{rd}$ steps in Figure 1 because these operations can be completed on ESP32. Since it is a software implementation, hashing will be slower than the hardware-accelerated version, but it will help us to avoid sending redundant HTTP requests. We compared these two approaches in Table 1 using *saveTemperature* smart contract function. The native hash row describes the Keccak-256 software implementation on the device, whereas the Web3 hash row shows the time spent using Web3 API hashing.

**Table 1**. Average times in milliseconds (ms) for sending a transaction.

| | PC | | ESP32 | |
|---|---|---|---|---|
| Method | Network Time (ms) | Total Time (ms) | Network Time (ms) | Total Time (ms) |
| Native Hash | 151.37 | 152.87 | 268.30 | 319.67 |
| Web3 Hash | 479.83 | 481.50 | 739.17 | 792.00 |

The flowchart for transaction send operation, including the introduced improvements, is represented in Figure 2. The left side of the dotted line represents the blockchain network, whereas the right side represents the ESP32 device and operations that take place on ESP32. As one can see, the nonce value is obtained from the blockchain network at the beginning and incremented on the device in each transaction send iteration.

Each running time has been calculated by averaging 30 consecutive successful send transaction requests for more precision directly by coding without using any packet analyzer tool. Results are shown in Table 1. The total time column covers the time from beginning to receiving the resulting transaction hash. Network time means time spent for sending and receiving HTTP requests. Network time also includes the time spent in the blockchain network, which contains broadcasting, validating, and publishing times of the transaction between

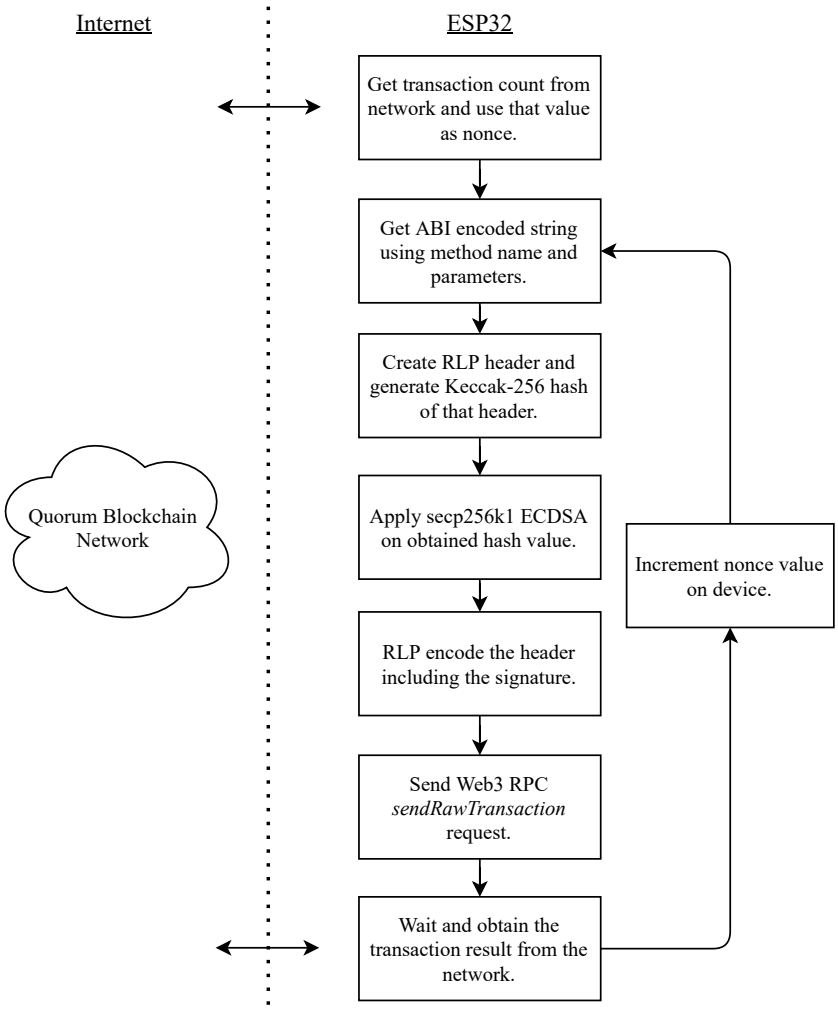


**Figure 2**. Flowchart for transaction send operation.

nodes. The difference between total time and network time shows the actual time spent in the ESP32 and PC. This difference contains all RLP, ABI, and JSON encoding-decoding, ECDSA, and Keccak-256 operations. It is basically all operations happening in the device.

As we can see in Table 1, if we use software implementation of the hash function in ESP32, completing a transaction send operation becomes approximately 2.5 times faster for ESP32 since we avoid extra HTTP packets for hashing. Network times might change because of unsteady network latency and server location; however, the performance increase ratio stays the same.

If we exclude the network time, the time spent for transaction send operation in ESP32 is 51.37 ms on average. For PC, this duration is 1.5 ms. PC implementation is faster as expected since ESP32 has lower processing power than regular PCs. PC can handle encoding-decoding and ECDSA operations very quickly, which makes a big difference. It is also important to note that PC network time is much lower because PC has strong networking and processing capabilities compared to ESP32, which affects network performance. PC implementation was given to show the baseline of how fast the transaction can be. The PC we used in this work

has an Intel Core i7 2.20 GHz processor with 16GB of random access memory (RAM) and a running Windows 10 operating system.

After we obtained the performance results of sending a transaction with ESP32, we also tested a call request using *getLastTemperatures* function. Call requests on a blockchain network do not require transaction signing operation. The average of 30 consecutive successful call requests is 341.4 ms. If the network time is excluded, time spent in the ESP32 device for a call request is only 3 ms on average.

As for packet sizes, a single transaction-send-request using *saveTemperature* sends 1207 bytes to the network and receives 834 bytes from the network. After the improvements, the total data sent is reduced to 457 bytes, and the total data received from the network is reduced to 224 bytes. These sizes include HTTP header sizes. As a result, there is a 66.63% reduction in communication costs.

### 4.2. Performance of elliptic curve signing

For ECC options in ESP32, we compared Bitcoin Core's libsecp256k1 C library with Mbed transport layer security's (TLS) hardware accelerated ECC implementation on ESP32. Bitcoin Core's libsecp256k1 achieved a better performance than Mbed TLS's ECDSA implementation on ESP32, as we can see in Table 2. Since it is faster, the libsecp256k1 library is used in the experiments in Table 1.

**Table 2**. Average times in milliseconds (ms) for ECDSA.

| ECDSA Library | PC Time (ms) | ESP32 Time (ms) |
|---|---|---|
| libsecp256k1 | 1.14 | 45.17 |
| Mbed TLS | 9.80 | 569.77 |

We can also deduct that most of the computing time in the ESP32 device is consumed by ECDSA operation with 45.17 ms of 51.37 ms on average. The remaining parts like hashing and encoding-decoding operations have individually lesser impacts compared to ECDSA. A previous study showed that ECDSA with secp256r1 curve takes 250 ms on average on ESP32, which is much higher than the results we obtained [34].

### 4.3. Quality of private key generation

For generating a private key, we used ESP32's built-in random function. This function returns a 32-bit number on each execution. It is indicated in the ESP32 technical reference manual that this RNG function can be used in cryptographic applications. In addition to that, we tested ESP32 RNG's quality using NIST statistical test suite (NIST STS) [35]. Testing the quality of random numbers is a hard task since there is no certainly approved method to calculate that. However, NIST STS gives us a good overview of the statistical quality of the output and its usability in cryptographic applications.

We generated 16 million bits of data using ESP32 RNG, which is enough for NIST STS to arrive at a conclusion. These 16 million bits consist of 32 different 500,000 bits long sequences. The NIST STS results of these produced sequences are shown in Table 3. Proportion means the ratio of the successful ones in these 32 sequences. Overall, ESP32 RNG achieves good proportions and p-values in NIST STS. As described in NIST STS documentation [35], to have randomized results, p-values must be greater than the $\alpha$ value, and $\alpha$ can be between 0.001 and 0.01, so from Table 3, all the tests except the Rank test satisfy this rule. The proportion value in the table indicates the proportion of the sequences passing the p-value test. The only failed test is the Rank test, where only 23 of 32 sequences can pass.

**Table 3**. NIST STS result for ESP32 RNG.

| Test | P-value | Proportion |
|---|---|---|
| Frequency | 0.253551 | 1.000000 |
| Block frequency | 0.911413 | 1.000000 |
| Cumulative sums | 0.276401 | 1.000000 |
| Runs | 0.017912 | 1.000000 |
| Longest run | 0.534146 | 1.000000 |
| Rank | 0.000000 | 0.718750 |
| FFT | 0.949602 | 1.000000 |
| Non overlapping template | 0.422745 | 0.989443 |
| Overlapping template | 0.976060 | 1.000000 |
| Universal | 0.213309 | 1.000000 |
| Approximate entropy | 0.066882 | 1.000000 |
| Random excursions | 0.208355 | 1.000000 |
| Random excursions Variant | 0.226062 | 1.000000 |
| Serial | 0.539435 | 1.000000 |
| Linear complexity | 0.911413 | 1.000000 |

Since the Rank test of NIST STS is failed on the sampled random data from ESP32, we applied a randomness extraction method to the RNG output using SHA-256 [36]. There are different methods for randomness extraction to improve the randomness quality. We picked SHA-256 because it is implemented in ESP32 with hardware acceleration. SHA-256 is applied to every 512 bits of data, so we can extract more entropy while reducing the size of the data by half. NIST STS results of this approach are shown in Table 4, where all test results in the suite are successful (described in the previous paragraph for Table 3).

### 4.4. Hashing performance

We calculated average running times for our hashing options on ESP32, which are shown in Table 5. These values are calculated by averaging running times of 100 consecutive hash operations. It is important to note that SHA-256 is a different hashing algorithm than Keccak-256. We only use it to compare the software implemented Keccak-256 function with ESP32's hardware-accelerated hashing.

ESP32's hardware-accelerated SHA-256 function has better performance as expected. Since we have to use Keccak-256 for the Quorum blockchain, we used the software implementation of it. This software implemented Keccak-256 function takes 0.20 ms for one operation on ESP32. Another work calculated SHA3-256 performance on ESP32, where they obtained 0.18 ms on average [37]. Although that value is very similar to our experiments, SHA3-256 slightly differs from Keccak-256 on implementation. On PC, the same Keccak-256 software implementation is completed in 0.03 ms on average.

### 4.5. Power consumption

ESP32 offers different power modes that are explained in the technical document in detail. In active mode, ESP32's radio chip is working, which enables Wi-Fi and Bluetooth technologies. Since we are actively using Wi-Fi, our device is always in active mode. In active mode, ESP32 specification indicates that while using

**Table 4**. NIST STS result for entropy extracted ESP32 RNG.

| Test | P-value | Proportion |
|---|---|---|
| Frequency | 0.739918 | 0.968750 |
| Block frequency | 0.534146 | 1.000000 |
| Cumulative sums | 0.519585 | 1.000000 |
| Runs | 0.468595 | 1.000000 |
| Longest run | 0.534146 | 1.000000 |
| Rank | 0.739918 | 1.000000 |
| FFT | 0.911413 | 1.000000 |
| Non overlapping template | 0.433924 | 0.990287 |
| Overlapping template | 0.178278 | 0.968750 |
| Universal | 0.299251 | 1.000000 |
| Approximate entropy | 0.949602 | 0.968750 |
| Random excursions | 0.327001 | 1.000000 |
| Random excursions variant | 0.291733 | 0.992063 |
| Serial | 0.437843 | 0.968750 |
| Linear complexity | 0.671779 | 1.000000 |

**Table 5**. Average running times in milliseconds (ms) for hashing.

| Hash Library | ESP32 Time (ms) |
|---|---|
| SHA-256 (hardware accelerated) | 0.07 |
| Keccak-256 (software) | 0.20 |

Wi-Fi transmission with a 3.3 V supply, power consumption is between 180 mA and 240 mA. After the data is received, power consumption values drop between 95 mA and 100 mA. In active mode, while the Wi-Fi is enabled, ESP32 switches between active mode and modem-sleep mode. So, power consumption decreases while the device is in modem-sleep mode. ESP32-WROOM-32 used in this work has power consumption values between 30 mA and 68 mA in modem-sleep mode.

We used a programmable single-channel power supply (OWON P4305) for measuring current values on ESP32. Measured values are obtained from the power supply device via the RS323 interface with a very low response time. The current drawn by ESP32 was followed instantly with a Python script on the PC. We tested a set of cases for power consumption values of ESP32, and results are shown in Table 6. When sending a transaction, we have the highest power consumption average with 102 mA because Wi-Fi communication is needed for that task.

For ECDSA methods, libsecp256k1 and Mbed TLS are compared. These ECDSA methods continuously work in a loop for calculating their power consumption. The average power consumption of libsecp256k1 library with 64 mA is better than Mbed TLS, which works on 67 mA on average. Previous work also shows that optimized ECC curves have better energy consumption values than RSA [38].

Hash functions, Keccak-256 and SHA-256, are also compared. The software implementation of Keccak-256 has more power consumption than hardware-accelerated SHA-256. If there will be a Keccak-256 hardware implementation in ESP32 devices, using it will provide better power consumption values.

**Table 6**. Power consumption values in mA for ESP32.

| | Mean (mA) | Median (mA) | Min (mA) | Max (mA) |
|---|---|---|---|---|
| Sending transaction | 102 | 98 | 47 | 266 |
| libsecp256k1 Sign | 64 | 64 | 47 | 65 |
| Mbed TLS Sign | 67 | 64 | 47 | 69 |
| Keccak-256 Hash | 73 | 73 | 47 | 73 |
| SHA-256 Hash | 70 | 70 | 47 | 73 |

We also analyzed the power consumption of ESP32 while sending a sequence of transactions to the Quorum blockchain network. As we can see in Figure 3, the current value increases above 250 mA while sending a transaction. Also, we can see other small peaks between 50 mA and 150 mA when RLP encoding, ECDSA, and hashing operations happen.

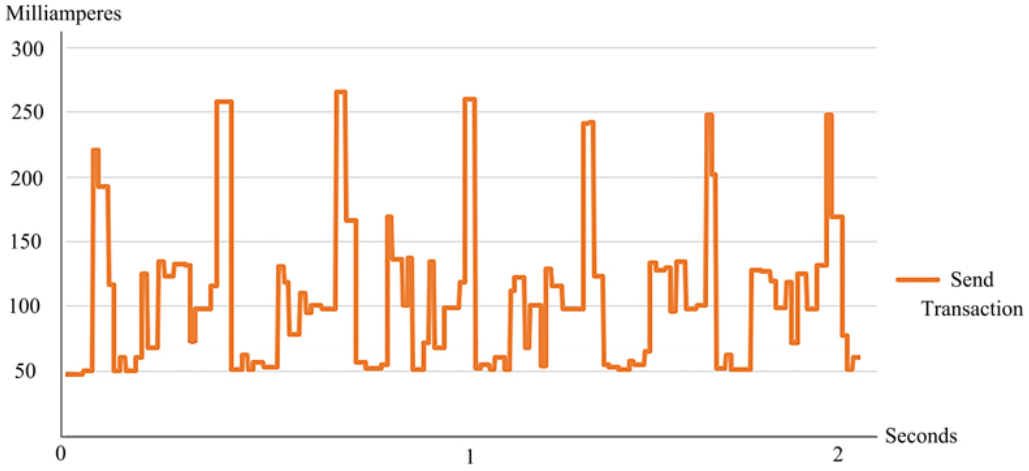


**Figure 3**. Network flow of sending a raw transaction.

## 5. Conclusion and future works

As a result of this study, we showed that ESP32 devices have sufficient capabilities as clients in a blockchain network. They are low-power and low-cost, which makes them more suitable for certain tasks. We also made performance improvements on network traffic and operations occurring in the device. These improvements increase ESP32's performance when sending transactions and making smart contract calls to the blockchain network. Transaction send operation becomes 2.5 times faster, and communication cost is reduced by 66.63%. We also obtained low and reasonable power consumption values while ESP32 communicates with the blockchain network.

As an additional improvement, the secure boot and flash encryption capabilities of ESP32 can be examined in detail in later works. These functionalities help us to store verified and encrypted flash data and private keys on ESP32. In the future, when a hardware-accelerated version of Keccak-256 is implemented, it will be faster than the software implementation that we used on ESP32. This hardware-accelerated version will improve the power consumption of the device. Also, the library we implemented can be expanded with upcoming

functionalities of blockchain technology. Furthermore, the lightchain [39] blockchain system can be used for future implementations.

## References


[1] Li S, Xu LD, Zhao S. The internet of things: a survey. Information Systems Frontiers 2015; 17 (2) : 243-259. doi: 10.1007/s10796-014-9492-7

[2] Kshetri N. Can blockchain strengthen the Internet of things?. IT Professional 2017; 19 (4) : 68-72. doi: 10.1109/mitp.2017.3051335

[3] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. 2008.

[4] Yaga D, Mell P, Roby N, Scarfone K. Blockchain technology overview. 2018. doi: 10.6028/nist.ir.8202

[5] Buterin V. A next-generation smart contract and decentralized application platform. 2014.

[6] Wust K, Gervais A. Do you need a blockchain? In: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). 2018. pp. 45-54. doi: 10.1109/cvcbt.2018.00011

[7] Sankar LS, Sindhu M, Sethumadhavan M. Survey of consensus protocols on blockchain applications. In: 4th International Conference on Advanced Computing and Communication Systems (ICACCS); 2017. doi:10.1109/icaccs.2017.8014672

[8] Huang D, Ma X, Zhang S. Performance analysis of the raft consensus algorithm for private blockchains. IEEE Transactions on Systems, Man, and Cybernetics: Systems 2020; 50 (1) : 172-181. doi: 10.1109/tsmc.2019.2895471

[9] H. Moniz. The Istanbul BFT consensus algorithm. 2020. arXiv: 2002.03613 [cs.DC].

[10] Nadir RM. Comparative study of permissioned blockchain solutions for enterprises. In: 2019 International Conference on Innovative Computing (ICIC); Pakistan; 2019. pp. 1-6. doi: 10.1109/icic48496.2019.8966735

[11] Maier A, Sharp A, Vagapov Y. Comparative analysis and practical implementation of the ESP32 microcontroller module for the Internet of things. In: 2017 Internet Technologies and Applications (ITA); Wrexham; 2017. pp. 143-148. doi: 10.1109/itecha.2017.8101926

[12] Lin J, Shen Z, Zhang A, Chai Y. Blockchain and IoT based food traceability for smart agriculture. In: Proceedings of the 3rd International Conference on Crowd Science and Engineering (ICCSE'18); Singapore; 2018. pp. 1-6. doi: 10.1145/3265689.3265692

[13] Cha S-C, Chen J-F, Su C, Yeh K-H. A blockchain connected gateway for BLE-Based devices in the Internet of things. IEEE Access 2018; 6 : 24639-24649. doi: 10.1109/access.2018.2799942

[14] Nayak S, Narendra NC, Shukla A, Kempf J. Saranyu: Using smart contracts and blockchain for cloud tenant management. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD); San Francisco; 2018. doi: 10.1109/cloud.2018.00121

[15] Novo O. Blockchain meets IoT: An architecture for scalable access management in IoT. IEEE Internet of Things Journal 2018; 5 (2) : 1184-1195. doi: 10.1109/jiot.2018.2812239

[16] Ma M, Shi G, Li F. Privacy-Oriented Blockchain-Based distributed key management architecture for hierarchical access control in the IoT scenario. IEEE Access 2019; 7: 34045-34059. doi: 10.1109/access.2019.2904042

[17] Chen J, Micali S. Algorand: A secure and efficient distributed ledger. Theoretical Computer Science 2019; 777: 155-183. doi: 10.1016/j.tcs.2019.02.001

[18] Ghosh D, Agrawal A, Prakash N, Goyal P. Smart saline level monitoring system using ESP32 and MQTT-S. In: 2018 IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom); Ostrava; 2018. pp. 1-5. doi: 10.1109/healthcom.2018.8531172

[19] Abdullah AH, Sudin S, Ajit MIM, et al. Development of ESP32-based Wi-Fi electronic nose system for monitoring LPG leakage at gas cylinder refurbish plant. In: 2018 International Conference on Computational Approach in Smart Systems Design and Applications (ICASSDA); Kuching; 2018. pp. 1-5. doi: 10.1109/icassda.2018.8477594

[20] Rai P, Rehman M. ESP32 Based smart surveillance system. In: 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET); Sukkur; 2019. pp. 1-3. doi: 10.1109/icomet.2019.8673463

[21] Allafi I, Iqbal T. Design and implementation of a low cost web server using ESP32 for real-time photovoltaic system monitoring. In: 2017 IEEE Electrical Power and Energy Conference (EPEC); Saskatoon; 2017. pp. 1-5. doi: 10.1109/epec.2017.8286184

[22] Baliga A, Subhod I, Kamat P, Chatterjee S. Performance evaluation of the Quorum blockchain platform. 2018. arXiv: 1809.03421 [cs.CR].

[23] Polge J, Robert J, Le Traon Y. Permissioned blockchain frameworks in the industry: A comparison. ICT Express 2021: 7 (2) : 229-233. doi:10.1016/j.icte.2020.09.002

[24] Enterprise Ethereum Alliance: 'EEA Client Specification v7', 2021

[25] Nasir Q, Qasse IA, Abu Talib M, Nassif AB. Performance analysis of Hyperledger Fabric platforms. Security and Communication Networks 2018; 2018: 1-14. doi: 10.1155/2018/3976093

[26] Bertoni G, Daemen J, Peeters M, Assche GV. The making of KECCAK. Cryptologia 2014; 38 (1) : 26-60. doi: 10.1080/01611194.2013.856818

[27] Barker E, Roginsky A. Transitioning the use of cryptographic algorithms and key lengths (No. NIST Special Publication (SP) 800-131A Rev. 2). National Institute of Standards and Technology 2019; doi:10.6028/NIST.SP.800-131Ar2

[28] NTT Information Sharing Platform Laboratories, NTT Corporation. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters. 2010.

[29] Johnson D, Menezes A, Vanstone S. The elliptic curve digital signature algorithm (ECDSA). International Journal of Information Security 2001; 1 (1) : 36-63. doi: 10.1007/s102070100002

[30] Bafandehkar M, Yasin SM, Mahmod R, Hanapi ZM. Comparison of ECC and RSA algorithm in resource constrained devices. In: 2013 International Conference on IT Convergence and Security (ICITCS); Macao; 2013. pp. 1-3. doi: 10.1109/icitcs.2013.6717816

[31] Turan MS, Barker E, Kelsey J, McKay KA, Baish ML et al. Recommendation for the entropy sources used for random bit generation. NIST Special Publication 800-90B Second Draft. Gaithersburg, MD, USA: NIST, 2018. doi: 10.6028/NIST.SP.800-90B

[32] Kösemen C, Dalkiliç G. Designing a random number generator for secure communication with WISP. In: Proceedings of the International Conference on Compute and Data Analysis (ICCDA'17); New York; 2017. pp. 289-292. doi: 10.1145/3093241.3093285

[33] Polge J, Robert J, Le Traon Y. Permissioned blockchain frameworks in the industry: a comparison. ICT Express 2020. doi: 10.1016/j.icte.2020.09.002

[34] Pearson B, Luo L, Zhang Y, Dey R, Ling Z et al. On misconception of hardware and cost in IoT security and privacy. In: 2019 IEEE International Conference on Communications (ICC); 2019. doi: 10.1109/icc.2019.8761062

[35] Bassham LE, Rukhin AL, Soto J, Nechvatal JR, Smid ME et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2010.

[36] Stipčević M, Koç ÇK. True random number generators. Open Problems in Mathematics and Computational Science 2014, Springer; 275-315. doi: 10.1007/978-3-319-10683-0_12

[37] Sarker VK, Gia TN, Tenhunen H, Westerlund T. Lightweight security algorithms for resource-constrained IoT-based sensor nodes. In: ICC 2020 - 2020 IEEE International Conference on Communications (ICC); Ireland; 2020. pp. 1-7. doi: 10.1109/icc40277.2020.9149359

[38] Suarez-Albela M, Fernandez-Carames TM, Fraga-Lamas P, Castedo L. A practical performance comparison of ECC and RSA for resource-constrained IoT devices. In: 2018 Global Internet of Things Summit (GIoTS); Spain; 2018. pp. 1-6. doi: 10.1109/giots.2018.8534575

[39] Hassanzadeh-Nazarabadi Y, Küpçü A, Özkasap Ö. LightChain: Scalable DHT-based blockchain. IEEE Transactions on Parallel and Distributed Systems 2021; 32 (10) : 2582-2593. doi: 10.1109/TPDS.2021.3071176